
Technical Description of TOBI Interface D (TiD)

Release 0.3.0.0

Christian Breitwieser

May 01, 2015

Contents

1	Introduction	2
1.1	License	2
1.2	Contact	2
2	Design principle	2
3	Connection principle	2
4	TiD Message	2
4.1	Structure	2
	Example	3
4.2	Version	3
4.3	Description	3
4.4	Block	3
4.5	Family	3
4.6	Event	4
4.7	Absolute	4
4.8	Relative	4
4.9	Source	5
4.10	Value	5
5	TiD Server	5
6	TiD client	5
7	TiD Architecture	5
8	TiD Performance Tuning	6
8.1	Unix/Linux:	6
8.2	Windows:	6
9	TiD Performance Tests	7
	Bibliography	7

1 Introduction

TiD (TOBI Interface D) describes a protocol to distribute events and markers used for brain-computer interface (BCI) purposes. It is based on a client–server principle, whereby the server acts as a distributor, dispatching incoming messages to every other connected client. The principle is somehow similar to a bus.

TiD is mainly intended to facilitate event distribution in a network environment.

It is not intended to replace any direct function calls within an established system. A direct function call will always be the fastest way to exchange information within one and the same process, so TiD will never be a replacement for that.

1.1 License

The TiD library is licensed under the [LGPLv3](#).

1.2 Contact

For further information please contact c.breitwieser@tugraz.at.

2 Design principle

TiD is designed to distribute BCI events to multiple clients, based on a client–server system. If a client creates an event, this event is sent to the TiD server, which dispatches it to all connected clients. To ensure proper timing every TiD message is equipped with a block number corresponding to the respective block the client was processing when the event occurred. Additional relative and absolute timestamps (in microseconds) are included into a TiD message, to provide inter-frame accuracy (assign an event to samples inside a frame). If a client is not involved in data processing (and is therefore not aware of the actual block number), the actual frame number is inserted by the TiD server before distributing the event. Therefore a TiD server needs some communication with the data acquisition system (e.g. TOBI Signal Server using TiA [\[TiA-Doc\]](#) [\[TiA-IEEE\]](#)).

To ensure events being always synchronous with the data, every processing module/step should forward the frame number of the actual data.

3 Connection principle

A TiD server has to provide a TCP port on which clients can establish a connection. Using a TCP acceptor, this connection is then bound to a dedicated port on the TiD Server. Each client gets its own connection to the server.

Via this connection TiD messages are sent to the server and distributed to all other clients.

Some important remarks: 1. The messages are encoded in UTF-8 2. All characters are case sensitive!

4 TiD Message

4.1 Structure

Each message, which is sent from the client to the server or vice versa, simply contains an XML TiD message string as follows:

Example

```
1 <tid version="0.3.0.0"  
2 description="beep"  
3 block="1732"  
4 family="biosig"  
5 event="785"  
6 absolute="1330691458,821096"  
7 relative="34687,761248"  
8 source="P300 detector"  
9 value="3,14159"/>
```

The TiD message has some mandatory and some optional attributes. These attributes and their intention are described in succession.

4.2 Version

Mandatory

A version attribute to avoid version incompatibilities during message processing.

The versions follow the \$CURRENT.\$REVISION.\$MINOR.\$BUGFIX schema and the following rules:

- If any “big” new features have get added, resulting in heavy interface changes, increment \$CURRENT, and set all lower fields to “0”.
- If any interfaces have been added, removed, or changed since the last update, increment \$REVISION, and set all lower fields to “0”.
- If the library source code has changed at all since the last update then increment \$MINOR and set \$BUGFIX to “0”
- If any bugs have got fixed since the last public release, then increment \$BUGFIX.

4.3 Description

Mandatory

A short human readable description of the event. Mainly intended to distinguish events more easily during manual event inspection (e.g., “beep”, “cue left”, “flash row 5”, ...)

4.4 Block

The data sample (or potentially block), the event belongs to

Dependent on the data acquisition hardware, a group (or block) of samples might get acquired together (e.g., done by the g.USBamp – <http://www.gtect.at>).

In such a case, the data acquisition source is acquiring data with a defined sampling rate, but delivers the data over an API in data blocks. For example, the sampling rate is 500 Hz and the acquisition system provides data in blocks of 10 samples. Thus, every block contains 10 samples and the blocks are delivered with a rate of 50 blocks per second.

4.5 Family

Mandatory

The family can be seen as a parent group, the event belongs to. For example, the biosig project (<http://biosig.sf.net>) already defines a big amount of unique events for different areas (<http://sourceforge.net/p/biosig/code/ci/master/tree/biosig4matlab/doc/eventcodes.txt>).

In a common environment, all events might usually be from the same family, so the occurrence of an event clash or event misinterpretation is unlikely. However, to avoid such issues, an event family can get defined in TiD.

That way, a client can choose to react only on events from a known family. So even heterogeneous systems with different event sources become possible.

Current event family definitions available in TiD:

- biosig (*to support events defined by the biosig project*)
- custom (*to support custom events, not being defined anywhere*)

The number of event families could also get extended in future to further support events from other prominent BCI systems, as mentioned below.

Potential other families for the future:

- BCI2000
- OpenViBE

4.6 Event

Mandatory

In TiD an event is treated as a an occurrence of a unique happening. Out of this reasons, events are currently assigned to an event code, inspired by the biosig project, mentioned above.

The event is just an integer value. This brings the advantage of fast event processing without the need to parse event strings.

In reverse, every event type (for example a “beep”) needs to get an event code assigned. It is a valid point of discussion to provide simple strings as a potential event as well. The current implementation only provides the processing of integer values.

4.7 Absolute

Mandatory

An absolute timestamp in microseconds since 1970-01-01 00:00:00. Internally, the “gettimeofday()” method is utilized to obtain the time.

With this value, a proper synchronization of events becomes possible. Please note: TiD does not offer a clock synchronization mechanism itself. The system clocks of different computers need to get synchronized by other systems like NTP (network time protocol) or PTP (precision time protocol).

4.8 Relative

Mandatory

This timestamp provides a timing value in microseconds, relative to a definable time point. Currently, the start of the TiD server is used as a reference.

That way, a customizable point of time can get used as “0” and a dedicated timeserver as stratum.

Furthermore, the relative timestamp is also allowed to get reset at any point of time, for example to trace the processing pipeline. By resetting the relative timestamp when the event is created and reading out its value when the event is finally processed, tracing becomes easily possible, especially on local systems.

4.9 Source

Optional

The source provides an optional field to specify the event origin like a P300 speller, etc. It is a simple descriptive string, which could be used to process events only from a defined source.

4.10 Value

Optional

The value is an optional floating point number, defining a potential value related to the event. For example, “BeepWithFrequency” as event and “1500” as value.

A more generic value typing system (e.g., to also allow sting) would be a meaningful extension. However, it was just not needed up to now.

5 TiD Server

The server is responsible for receiving and dispatching of incoming events.

Events without a frame number and/or timestamp get the actual values from the data acquisition. An unknown block number is marked with “-1”. Therefore it needs to have access to the data acquisition system. It is suggested to include a TiD server into the data acquisition like the TOBI SignalServer.

It provides the possibility to gather events received from connected clients for saving purposes. These events are stored and can get saved elsewhere (e.g. in a simple text file or a .gdf file).

All TiD connections are running in separated threads to facilitate fast event processing, especially on nowadays multi-core computer systems. In case of an error, the connection and the receiver thread get closed.

The TiD Server is listening at port 9001 by default. To decrease delivery latency, Nagle’s algorithm is deactivated (TCP_NODELAY flag is set).

6 TiD client

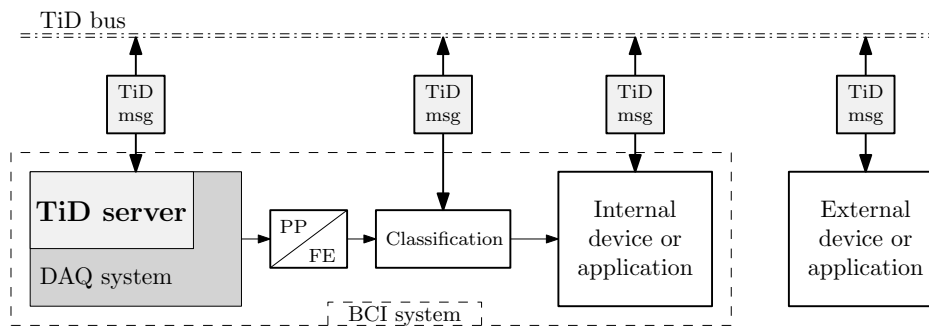
A TiD client can send and receive TiD messages using the connection to the TiD server. All communication is handled via this connection.

The client is responsible by itself for processing the TiD messages. It can simply ignore messages it does not care for.

Every client, also processing incoming data (e.g. biosignals, classification results) has to include the according frame number into an outgoing TiD message. If a client is not in touch with the BCI processing chain (pre-processing, feature extraction, classification, fusion, shared control), e.g. a P300 speller just sending events, has to leave the frame number blank, the actual value will be inserted by the server before dispatching.

7 TiD Architecture

As mentioned at the beginning, TiD is acting in a bus oriented manner. The following figure illustrates the principle:



8 TiD Performance Tuning

8.1 Unix/Linux:

In a Unix/Linux environment it is suggested to use a custom kernel.

Potential tuning options would be the modification of the timer frequency: frequency to quickly react on events

```

Symbol: HZ_1000 [=y]
Location:
-> Processor type and features
-> Timer frequency (<choice> [=y])

```

And the modification of the preemption model:

```

Symbol: PREEMPT_VOLUNTARY [=n]
Location:
-> Processor type and features
-> Preemption Model (<choice> [=y])

```

As suggested by RedHat (<http://developerblog.redhat.com/2015/02/11/low-latency-performance-tuning-rhel-7/>, <https://access.redhat.com/videos/898583>), adjusting system settings can also affect network latency.

According to the RedHat sources mentioned above following additional settings for sysctl.conf should be meaningful

```

net.core.wmem_max=12582912
net.core.rmem_max=12582912
net.ipv4.tcp_rmem= 10240 87380 12582912
net.ipv4.tcp_wmem= 10240 87380 12582912

net.core.busy_read=50
net.core.busy_pull=50
net.ipv4.tcp_fastopen=1
kernel.numa_balancing=0

kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 10000000
vm.dirty_ratio = 10
vm.dirty_background_ratio = 3
vm.swappiness=10
transparent_hugepage=never
kernel.sched_migration_cost_ns = 5000000

```

8.2 Windows:

The windows scheduler acts, depending on operating system and host related configuration, in defined time slots, also often called “quantum”. Such a quantum can get interpreted as a guaranteed amount of time for a certain

thread/process.

Such quantumts can get interrupted by operating system interrupts. As discussed in “The Windows Timestamp Project” (<http://www.windowstimestamp.com/description>), Windows offers some undocumented functions to re-configure the interrupt time resolution (like “*NtSetTimerResolution*”).

The code within the performance tests of the TiD library already contains some demo code, setting the scheduler to the maximum timer granularity, resulting in potential smaller delays, even in higher workload situations.

Unfortunately Microsoft does not offer as many latency tuning options as Linux. Some additional options, which are unfortunately (partly) only available on Windows Server OS are described on the following site: <https://technet.microsoft.com/en-us/library/jj574151.aspx>

Thus, it is suggested to disable power saving options of the network adapter and further also disable power saving options from the CPU (like C-states).

9 TiD Performance Tests

The TiD library offers automated performance testing methods. These methods are available in the “tid_tests” sub-project.

Within these tests, the individual latency then sending/receiving or dispatching a defined amount of messages for variable TiD message lengths can get obtained. Furthermore, the testing of the localhost as well as the network latency is possible.

This sub-project also offers Matlab script to load and analyze the test results. Histograms show the overall network latency. The individual transfer function for the individual plots were calculated as well, presenting the influence of the jitter when averaging data series, aligned on the basis of TiD events.

That way, every TiD user can determine the network environment latency conditions on his/her own to get a feeling for the potential limitations.

References

- [TiA-Doc] C. Breitwieser and C. Eibel, “TiA – Documentation of TOBI Interface A”, ArXiv e-prints, Mar. 2011. (<http://arxiv.org/abs/1103.4717>)
- [TiA-IEEE] C. Breitwieser, I. Daly, C. Neuper, and G. R. M?ller-Putz, “Proposing a standardized protocol for raw biosignal transmission”, IEEE. Trans. Biomed. Eng., vol. 59, no. 3, pp. 852-859, 2012.